ESTUDAR E REALMENTE APRENDER DEVOPS & CLOUD

# Caderno de Exercícios

66 aulas de prática, do #01 ao #66

Linux · Git · Docker · Terraform · Kubernetes · 12 módulos · 73 questões · 2 desafios integradores

Autor Desconhecido · theknowledgeisle.com

Nome do aluno: Início: Término:

**Como usar este caderno**

Cada módulo reúne um bloco de aulas do curso, com a lista dos artigos que ele cobre. As perguntas conceituais pedem explicações com as suas palavras; os exercícios de código pedem programas: escreva, compile e execute cada um.

Tente sempre resolver sozinho antes de abrir o **Caderno de Respostas**. É no erro que o conhecimento fixa.

## Sumário


| Módulo | Tema | Aulas | Questões |
|---|---|---|---|
| MÓDULO 1 | Linux e o terminal | #01–#06 | Q1–Q10 |
| MÓDULO 2 | Git e colaboração | #07–#11 | Q11–Q17 |
| MÓDULO 3 | Containers | #12–#18 | Q18–Q23 |
| MÓDULO 4 | CI/CD | #19–#24 | Q24–Q28 |
| MÓDULO 5 | Infraestrutura como código | #25–#30 | Q29–Q35 |
| MÓDULO 6 | Observabilidade | #31–#34 | Q36–Q42 |
| MÓDULO 7 | AWS na prática | #35–#38 | Q43–Q48 |
| MÓDULO 8 | Kubernetes | #39–#41 | Q49–Q54 |
| MÓDULO 9 | Segurança, custos, resiliência e cultura | #42–#47 | Q55–Q61 |
| MÓDULO 10 | Capstone: o sistema completo | #48–#52 | Q62–Q64 |
| MÓDULO 11 | Outras nuvens e plataformas | #53–#60 | Q65–Q68 |
| MÓDULO 12 | IA em operações e extras | #61–#66 | Q69–Q73 |
| DESAFIOS | Desafios integradores | todas | D1–D2 |

Legenda: nas perguntas, `assim` indica código, nomes de tipos e comandos. As linhas pautadas são para a sua resposta; nos exercícios de código, use também o computador: **todo programa deve ser compilado e executado**.

MÓDULO 1

# Linux e o terminal

O terminal, o sistema de arquivos, permissões, processos e serviços, shell script e SSH.

### Artigos 01 a 06

**#01** O Terminal Não é o Inimigo
**#02** Navegando pelo Sistema de Arquivos com Confiança
**#03** Permissões, Usuários e Grupos no Linux
**#04** Processos, Serviços e o Comando `systemctl`
**#05** Shell Script do Zero
**#06** SSH: Conectando e Gerenciando Servidores Remotos

## Perguntas conceituais

**Q1** **Caminho absoluto e relativo**

Estando em `/home/ana/projetos`, para onde levam `cd ..`, `cd ../..`, `cd ~/docs`, `cd -` e `cd /var/log`? Por que um script agendado no cron deve usar caminhos absolutos?

**Q2** **Lendo permissões**

O que significa `-rwxr-x---` num arquivo, e qual o equivalente numérico? Por que `chmod 777` é quase sempre um erro, e o que o bit de execução significa num diretório?

**Q3** **Processos e sinais**

Qual a diferença entre `kill PID` e `kill -9 PID`? Por que o primeiro deve ser sempre tentado antes, e o que é um processo zumbi?

**Q4** **systemd**

O que é uma *unit* de serviço do systemd? O que `systemctl enable --now nginx` faz, e onde ver o *log* de um serviço que não sobe?

**Q5** **set -euo pipefail**

O que cada opção de `set -euo pipefail` faz no início de um *script* Bash, e que problema cada uma evita?

**Q6** **SSH com chaves**

Por que autenticar no servidor com chave SSH é mais seguro que com senha? Que duas linhas do `sshd_config` todo servidor exposto à internet deveria ter?

## Exercícios de código

**Q7** **Permissões na prática**

Num diretório temporário, crie um arquivo e um diretório, mostre as permissões criadas com o `umask` padrão, depois aplique `chmod 640` ao arquivo e `chmod 750` ao diretório, e use a forma simbólica (`u+x`, `g-w`) para mais dois ajustes. Mostre as permissões com `stat` depois de cada passo.

**Q8** **Processos em segundo plano e sinais**

Escreva um *script* que inicie um processo filho em segundo plano que trata o SIGTERM (com `trap`) para "limpar" antes de sair; o pai espera um pouco, envia `kill` (SIGTERM), espera o filho com `wait` e mostra o código de saída. Depois repita com um filho que **ignora** o SIGTERM e precisa do `kill -9`.

## Q9 Um script de backup com rotação

Escreva `backup.sh` com `set -euo pipefail`, que receba um diretório de origem e um de destino, valide os argumentos, crie um `.tar.gz` com a data no nome e apague os *backups* com mais de 7 dias, mantendo sempre os 3 mais recentes. Simule backups antigos com `touch -d` para testar a rotação.

## Q10 Analisando um log de acesso

Dado um trecho de *log* de acesso do nginx, use `awk`, `sort`, `uniq` e `grep` para mostrar: (a) as requisições por código de status; (b) os 2 IPs com mais requisições; (c) as URLs que deram erro 5xx; (d) o total de bytes enviados.

MÓDULO 2

# Git e colaboração

Git na prática, Git Flow, a primeira automação com GitHub Actions, branches protegidas e pull requests, Semantic Versioning e tags.

### Artigos 07 a 11


**#07** Git na Prática: Commits, Branches e Merges sem Medo
**#08** Trabalhando em Equipe com Git Flow
**#09** GitHub Actions: Sua Primeira Automação de CI/CD
**#10** Protegendo Branches e Revisando Código com Pull Requests
**#11** Semantic Versioning e Tags de Release


## Perguntas conceituais

**Q11** **As três áreas do Git**

Explique o diretório de trabalho, a *staging area* (índice) e o repositório. O que `git add -p` permite, e por que a *staging area* existe?

**Q12** **merge, rebase e o histórico**

Qual a diferença entre integrar uma *branch* com `git merge --no-ff` e com `git rebase`? Por que nunca se faz *rebase* de *commits* já publicados?

**Q13** **Branches protegidas**

Que regras de proteção a *branch* `main` de um projeto em equipe deve ter no GitHub, e por quê?

### Q14 Semantic Versioning

Dada a versão `2.4.1` de uma biblioteca, qual é a próxima versão em cada caso: (a) corrigiu um *bug* sem mudar a API; (b) acrescentou uma função nova; (c) removeu um parâmetro de uma função existente. O que significa `^2.4.1` num `composer.json` ou `package.json`?

## Exercícios de código

### Q15 Um fluxo com branches e merge --no-ff

Num repositório novo, crie a `main` com um *commit*, a *branch* `dev` a partir dela, uma *branch* `feature/busca` a partir da `dev` com dois *commits*, e integre-a na `dev` com `git merge --no-ff`. Mostre o grafo com `git log --graph --oneline` (sem os *hashes*) e liste os *commits* de *merge*.

### Q16 Uma tag anotada e o changelog

No mesmo estilo, crie três *commits* seguindo o Conventional Commits (`feat:`, `fix:`, `docs:`), crie a *tag* anotada `v1.0.0`, mais dois *commits* (um `feat:` e um `fix:`), e gere, com `git log`, a lista das mudanças desde a *tag*, agrupada por tipo. Com base nela, diga qual deve ser a próxima versão.

## Q17 Uma primeira pipeline do GitHub Actions

Escreva `.github/workflows/ci.yml` que rode em todo *push* e *pull request* para `main` e `dev`: faça o *checkout*, configure o PHP 8.5 com cache das dependências do Composer, rode `composer install`, o *lint* e os testes. Depois confira, com Python, que o arquivo é YAML válido e liste os gatilhos e os passos.

MÓDULO 3

# Containers

O que é um container, os primeiros containers, o Dockerfile, volumes e redes, Docker Compose, boas práticas de imagens e a publicação em registries.

### Artigos 12 a 18


**#12** O Que é um Container e Por Que Isso Mudou Tudo
**#13** Instalando e Rodando Seus Primeiros Containers
**#14** Escrevendo um Dockerfile do Zero
**#15** Docker Volumes e Redes: Persistência e Comunicação
**#16** Docker Compose: Orquestrando Múltiplos Serviços Localmente
**#17** Boas Práticas de Imagens: Leveza, Segurança e Camadas
**#18** Publicando Imagens no Docker Hub e GitHub Container Registry


## Perguntas conceituais

**Q18** **Container não é máquina virtual**

Qual a diferença entre um container e uma máquina virtual? Que recursos do kernel Linux tornam o container possível?

**Q19** **Imagem, container e camadas**

Qual a diferença entre imagem e container? Por que a ordem das instruções no Dockerfile afeta o tempo de *build*?

### Q20 Volume ou bind mount

Para o banco de dados de um container MySQL, por que usar um volume nomeado? E quando um *bind mount* (uma pasta do hospedeiro) é a escolha certa?

### Q21 Uma imagem mínima e segura

Cite quatro práticas que deixam uma imagem menor e mais segura, e por que rodar o processo como *root* dentro do container é um risco.

## Exercícios de código

### Q22 Um Dockerfile em múltiplos estágios para PHP

Escreva um `Dockerfile` para uma aplicação PHP com Composer: um estágio que instala as dependências sem as de desenvolvimento, e um estágio final com `php:8.5-apache`, a extensão `pdo_mysql`, o OPcache, o código, um usuário sem privilégios e o `DocumentRoot` apontando para `public/`. Escreva também o `.dockerignore`. Mostre o caminho equivalente sem Docker, num servidor Debian.

## Q23 Um docker-compose para a aplicação

Escreva um `compose.yaml` com três serviços: a aplicação PHP (construída do Dockerfile anterior, porta 8080), um MySQL 8.4 com volume nomeado e um *healthcheck*, e um Redis. A aplicação só deve subir depois que o MySQL estiver saudável, e a senha do banco deve vir de um arquivo `.env`. Confira com Python a estrutura: a ordem de dependência e os volumes declarados.

MÓDULO 4

# CI/CD

O que é CI/CD, pipelines de build, teste e deploy, testes no pipeline, variáveis e secrets, deploy automático e monitoramento do pipeline.

**Artigos 19 a 24**


#19 O Que é CI/CD e Por Que Sua Empresa Precisa Disso
#20 Pipeline com GitHub Actions: Build, Test e Deploy Automático
#21 Testes Automatizados no Pipeline: Qualidade sem Atrito
#22 Variáveis de Ambiente e Secrets em Pipelines
#23 Deploy Automático para Servidores com GitHub Actions
#24 Notificações, Logs e Monitoramento do Pipeline


## Perguntas conceituais

**Q24** **Integração, entrega e implantação contínuas**

Diferencie integração contínua, entrega contínua e implantação contínua. O que precisa ser verdade sobre os testes para que a implantação contínua seja segura?

**Q25** **Secrets no pipeline**

Como o GitHub Actions protege os *secrets*, e por que um *pull request* vindo de um *fork* não recebe os *secrets* do repositório? Que erro comum ainda vaza um segredo?

### Q26 O pipeline como portão

Num pipeline com as etapas *lint*, testes unitários, *build* da imagem, testes de integração, deploy em homologação e deploy em produção, em que ordem e com que dependências elas devem rodar? O que deve acontecer se os testes de integração falharem?

## Exercícios de código

### Q27 Um deploy por SSH com rollback

Escreva `deploy.sh`, que simule localmente a estratégia de *releases* com *symlink*: cria `releases/<versão>`, copia o código, troca o *symlink* `current` de forma atômica, roda uma verificação de saúde e, se ela falhar, volta o *symlink* para a versão anterior. Mantenha só as 3 últimas *releases*. Teste um deploy bom, um ruim e mais três bons.

**Q28** **Testes com serviços no pipeline**

Escreva um *job* do GitHub Actions que rode os testes de integração contra um MySQL 8.4 de verdade, usando `services` com *healthcheck*, e passe as credenciais por variáveis de ambiente, com a senha vinda de um *secret*. Confira com Python que o serviço tem *healthcheck* e que nenhuma senha está escrita em texto no arquivo.

MÓDULO 5

# Infraestrutura como código

Terraform (recursos na AWS, módulos e o state), Ansible e a combinação dos dois.

### Artigos 25 a 30


**#25** Introdução ao Terraform: Infraestrutura que Você Pode Versionar
**#26** Seus Primeiros Recursos na AWS com Terraform
**#27** Módulos no Terraform: Reusabilidade e Organização
**#28** State do Terraform: Entendendo o Arquivo Mais Crítico do Projeto
**#29** Ansible: Configurando Servidores de Forma Declarativa
**#30** Terraform + Ansible: Do Provisionamento à Configuração


## Perguntas conceituais

**Q29** **Declarativo ou imperativo**

O Terraform é **declarativo**; um *script* Bash que chama a CLI da AWS é **imperativo**. Explique a diferença com o exemplo "quero 3 servidores", e o que acontece em cada caso quando já existem 2.

**Q30** **O arquivo de state**

O que o `terraform.tfstate` guarda? Por que ele não deve ir para o Git, e por que um *backend* remoto com *lock* é obrigatório numa equipe?

**Q31** **Ler o plano antes de aplicar**

Num `terraform plan`, o que significam os símbolos `+`, `-`, `~` e `-/+`? Por que um `-/+` num banco de dados deve parar tudo?

**Q32** **Terraform e Ansible juntos**

Por que o curso combina Terraform e Ansible, em vez de usar só um deles? Qual é a fronteira entre os dois?

## Exercícios de código

**Q33** **Uma VPC e um servidor com Terraform**

Escreva o Terraform para criar, na AWS, uma VPC `10.0.0.0/16` com uma sub-rede pública, um *security group* que libere a porta 443 para a internet e a 22 só para o seu IP (recebido como variável), e uma instância `t3.micro` com Debian. Use variáveis, *tags* comuns e uma saída com o IP público.

## Q34 Uma política contra destruição no plano

O `terraform show -json plano` gera o plano em JSON. Escreva um *script* Python que leia esse JSON e **reprove** o plano (código de saída 1) se ele destruir ou recriar qualquer recurso dos tipos `aws_db_instance` ou `aws_s3_bucket`, listando as ações de todos os recursos. Teste com um plano de exemplo.

## Q35 Idempotência como no Ansible

O módulo `lineinfile` do Ansible garante que uma linha exista num arquivo, e só altera o arquivo se precisar, informando `changed` ou `ok`. Escreva em Bash a função `garantir_linha ARQUIVO REGEX LINHA`, com o mesmo comportamento: se uma linha casar com a expressão regular, ela é substituída pela nova (se for diferente); senão, a linha é acrescentada. Rode-a duas vezes sobre um `sshd_config` de exemplo e mostre que a segunda não muda nada.

MÓDULO 6

# Observabilidade

Métricas, logs e traces; Grafana, dashboards e alertas; rastreamento distribuído com OpenTelemetry; alertas inteligentes e resposta a incidentes.

**Artigos 31 a 34**


#31 Introdução ao Monitoramento: Métricas, Logs e Traces
#32 Grafana: Dashboards e Alertas que Fazem Sentido
#33 Rastreamento Distribuído com OpenTelemetry
#34 Alertas Inteligentes e Cultura de Resposta a Incidentes


## Perguntas conceituais

### Q36 Os três pilares

Qual pergunta cada pilar da observabilidade responde melhor: métricas, *logs* e *traces*? Dê um exemplo de uso de cada um durante um incidente de lentidão no *checkout*.

### Q37 Por que percentis, e não a média

Por que um painel deve mostrar a latência p95 ou p99, e não a média? Dê um exemplo numérico.

### Q38 SLO e orçamento de erro

O que são SLI, SLO e orçamento de erro? Se o SLO de disponibilidade do *checkout* é 99,9% por mês, quanto tempo de indisponibilidade o mês permite, e o que a equipe faz quando o orçamento acaba?

### Q39 Alertas que acordam alguém

Por que alertar sobre "CPU acima de 80%" costuma ser um alerta ruim, e "taxa de erros do *checkout* consumindo o orçamento de erro 14 vezes mais rápido que o normal por 1 hora" é um bom? O que é fadiga de alertas?

## Exercícios de código

### Q40 Percentis de latência

Dado um conjunto de 20 latências (em ms) de um *endpoint*, calcule a média, a mediana, o p95 e o p99 com Python, e mostre como uma única requisição muito lenta afeta a média e o p99, mas não a mediana.

### Q41 Orçamento de erro e burn rate

Escreva um *script* Python que, para um SLO de 99,9% num mês de 30 dias, calcule o orçamento de erro em minutos e em requisições (para 10 milhões de requisições no mês), e, dada a taxa de erros da última hora, a *burn rate* e em quantos dias o orçamento acabaria. Classifique em "página alguém" (burn rate ≥ 14,4), "abre ticket" (≥ 6) ou "normal".

**Q42** **Logs estruturados com correlação**

Configure o `logging` do Python para emitir uma linha JSON por evento, com data em ISO 8601 UTC, nível, mensagem, o nome do serviço e um `trace_id` que acompanha todos os *logs* de uma mesma requisição. Simule duas requisições, uma com erro, e mostre como filtrar os *logs* de uma delas.

MÓDULO 7

# AWS na prática

EC2, VPC e IAM; ECS e Lambda; RDS, ElastiCache e estratégias de dados; Route 53, CloudFront e ACM.

**Artigos 35 a 38**

**#35** AWS na Prática: EC2, VPC e IAM em Profundidade
**#36** ECS e Lambda: Containers e Serverless na AWS
**#37** RDS, ElastiCache e Estratégias de Dados na AWS
**#38** Route53, CloudFront e ACM: Rede e Entrega de Conteúdo

## Perguntas conceituais

### Q43 Sub-rede pública e privada

Numa VPC, o que torna uma sub-rede "pública" ou "privada"? Em qual delas ficam o *load balancer*, os servidores da aplicação e o banco RDS, e como os servidores privados baixam atualizações da internet?

### Q44 IAM e o menor privilégio

Por que a aplicação na EC2 deve usar uma **role** do IAM, e não chaves de acesso (`AWS_ACCESS_KEY_ID`) gravadas no servidor? O que é o princípio do menor privilégio numa política?

### Q45 ECS ou Lambda

Para cada parte da livraria, escolha entre ECS Fargate e Lambda: (a) a API principal, com tráfego constante; (b) o redimensionamento das capas enviadas; (c) um relatório que roda uma vez por noite e leva 40 minutos.

**Q46** **CDN e cache**

O que o CloudFront faz por um site com usuários no Brasil todo e servidor em São Paulo? Por que as URLs de arquivos estáticos costumam ter um *hash* no nome (`app.3f9a2c.css`)?

## Exercícios de código

**Q47** **Planejando sub-redes**

Com o módulo `ipaddress` do Python, divida a VPC `10.0.0.0/16` em sub-redes `/20` e reserve as seis primeiras: pública, aplicação e dados, em duas zonas de disponibilidade. Mostre, para cada uma, o intervalo, o número de endereços utilizáveis na AWS (que reserva 5 por sub-rede) e verifique que não há sobreposição.

**Q48** **Auditando uma política do IAM**

Escreva um *script* Python que analise uma política do IAM (JSON) e aponte: declarações com `"Action": "*"` ou serviço inteiro (`"s3:*"`), `"Resource": "*"` em ações de escrita, e ausência de condição em `iam:PassRole`. Teste com uma política ruim e uma boa.

MÓDULO 8

# Kubernetes

Orquestrando containers em escala, EKS e Kubernetes em produção: segurança, GitOps e deploys avançados.

**Artigos 39 a 41**

**#39** Introdução ao Kubernetes: Orquestrando Containers em Escala
**#40** EKS: Kubernetes Gerenciado na AWS
**#41** Kubernetes em Produção: Segurança, GitOps e Deploys Avançados

## Perguntas conceituais

**Q49** **Pod, Deployment e Service**

Explique o papel de um Pod, de um Deployment e de um Service. Por que nunca se cria um Pod diretamente em produção?

**Q50** **Liveness e readiness**

Qual a diferença entre a *liveness probe* e a *readiness probe*? Por que usar a mesma verificação, que testa o banco de dados, nas duas pode derrubar o sistema inteiro?

**Q51** **requests e limits**

O que são `requests` e `limits` de CPU e memória num container? O que acontece quando um container passa do limite de memória, e quando passa do de CPU?

**Q52** **GitOps**

O que é GitOps, e qual a diferença entre o pipeline "empurrar" (o CI roda `kubectl apply` no cluster) e o modelo "puxar" do Argo CD ou do Flux?

## Exercícios de código

**Q53** **Um Deployment pronto para produção**

Escreva os manifestos de um Deployment da API da livraria (3 réplicas, imagem com *tag* fixa, `requests` e `limits`, *readiness* e *liveness* em rotas diferentes, usuário sem privilégios e sistema de arquivos somente leitura) e do Service. Depois valide com Python: as probes usam rotas diferentes, todos os containers têm limite de memória e nenhuma imagem usa `latest`.

**Q54** **A conta do autoscaler**

O HPA (*Horizontal Pod Autoscaler*) calcula as réplicas desejadas com `desejadas = teto(atuais × métrica_atual / métrica_alvo)`, ignora variações dentro de uma tolerância de 10% e respeita os limites mínimo e máximo. Implemente o cálculo em Python e simule uma rampa de carga com alvo de 60% de CPU, mínimo de 2 e máximo de 10 réplicas.

MÓDULO 9

# Segurança, custos, resiliência e cultura

DevSecOps, compliance e auditoria, FinOps, chaos engineering, platform engineering e a cultura DevOps.

**Artigos 42 a 47**


#42 DevSecOps: Integrando Segurança em Todo o Pipeline
#43 Compliance e Auditoria: LGPD, SOC2 e Evidências Automatizadas
#44 Performance e FinOps: Otimizando Custo e Velocidade na Cloud
#45 Resiliência e Chaos Engineering
#46 Platform Engineering: Construindo a Plataforma Interna de Desenvolvimento
#47 Cultura DevOps: Maturidade, Postmortems e Melhoria Contínua


## Perguntas conceituais

**Q55** **Shift left**

O que significa "levar a segurança para a esquerda" num pipeline? Cite quatro verificações automáticas e em que etapa cada uma roda.

**Q56** **LGPD na infraestrutura**

Que obrigações da LGPD afetam diretamente decisões de infraestrutura e de operação? Cite três exemplos práticos.

**Q57** **Onde o dinheiro vaza**

Cite quatro fontes comuns de desperdício numa conta de nuvem e a correção de cada uma.

### Q58 Chaos engineering

O que é chaos engineering, e por que ele começa com uma hipótese e um "raio de explosão" pequeno? Dê um experimento para a livraria.

### Q59 Postmortem sem culpados

Por que um postmortem deve ser "sem culpados" (*blameless*)? Que seções ele precisa ter para gerar aprendizado, e não só um relato?

## Exercícios de código

### Q60 Varredura de segredos

Escreva um *script* Python que procure segredos em arquivos de texto com expressões regulares: chaves de acesso da AWS (`AKIA` seguido de 16 caracteres), chaves privadas PEM, *tokens* do GitHub (`ghp_`) e atribuições do tipo `senha = "..."` ou `password: ...`. Mostre arquivo e linha, mascarando o valor encontrado, e devolva código 1 se achar algo.

## Q61 Comparando preços de instâncias

Escreva um *script* Python que compare o custo mensal de 4 servidores ligados o mês inteiro, em três modelos: sob demanda (US$ 0,0832 por hora cada), plano de economia de 1 ano (37% de desconto) e uma combinação de 2 reservados com 2 *spot* (70% de desconto, mas podem ser interrompidos). Calcule também o custo de um ambiente de homologação de 2 servidores sob demanda ligado só em horário comercial (10 h por dia, 22 dias por mês).

MÓDULO 10

# Capstone: o sistema completo

Arquitetura, provisionamento, microsserviços, o pipeline completo e as operações em produção.

**Artigos 48 a 52**


**#48** Projeto Capstone: Arquitetura do Sistema Completo
**#49** Capstone: Provisionando a Infraestrutura Completa
**#50** Capstone: Implementando os Microsserviços
**#51** Capstone: Pipeline Completo de CI/CD
**#52** Capstone: Operações em Produção e Retrospectiva da Jornada


## Perguntas conceituais

**Q62** **Microsserviços: quando vale a pena**

O capstone divide o sistema em microsserviços. Que problemas eles resolvem, que custos trazem, e por que muitas equipes deveriam começar com um monólito bem modularizado?

**Q63** **Implantação gradual**

Explique os deploys *blue-green* e *canary*. Que métricas decidem se o *canary* avança ou volta?

## Exercícios de código

**Q64** **Decidindo um canary**

Escreva uma função Python que receba as contagens de requisições e erros e a latência p95 da versão estável e do *canary* num intervalo, e decida entre "avançar", "aguardar" (amostra pequena demais, menos de 500 requisições no *canary*) e "reverter" (taxa de erros do *canary* acima de 1,5 vez a da estável **e** acima de 0,5% em termos absolutos, ou p95 mais de 20% pior). Teste quatro cenários.

MÓDULO 11

# Outras nuvens e plataformas

Azure para quem conhece AWS, compute, storage e redes no Azure, AKS, Azure DevOps, GitLab CI/CD, GitLab self-hosted, Bitbucket e Gitea.

**Artigos 53 a 60**


**#53** Azure para Quem Já Conhece AWS
**#54** Compute, Storage e Redes no Azure
**#55** Azure Kubernetes Service (AKS)
**#56** Azure DevOps: Pipelines e Repositórios
**#57** GitLab CI/CD: A Alternativa Enterprise ao GitHub Actions
**#58** GitLab Self-Hosted: Soberania Total sobre Código e Pipelines
**#59** Bitbucket e o Ecossistema Atlassian
**#60** Gitea: Self-Hosted Leve para Times Menores


## Perguntas conceituais

### Q65 AWS para Azure

Para quem conhece a AWS, qual o equivalente no Azure de: VPC, EC2, S3, RDS para PostgreSQL, IAM *roles* para aplicações, EKS e CloudWatch? Que conceito do Azure não tem equivalente direto?

### Q66 GitLab CI e GitHub Actions

Compare a estrutura de um `.gitlab-ci.yml` com a de um *workflow* do GitHub Actions. O que são *stages* e o que a palavra-chave `needs` muda?

### Q67 Hospedar a própria forja

Quando faz sentido hospedar o próprio GitLab ou Gitea, em vez de usar o GitHub ou o GitLab.com? Que responsabilidades a equipe assume?

## Exercícios de código

### Q68 Um pipeline do GitLab com needs

Escreva um `.gitlab-ci.yml` com os *stages* `build`, `test` e `deploy`: um *job* de *build*, dois de teste (unitários e *lint*), em que o *lint* não depende do *build*, e um deploy manual só na *branch* `main`. Use Python para calcular a ordem real de execução a partir do `needs`.

MÓDULO 12

# IA em operações e extras

AIOps, pipelines e infraestrutura gerados com IA, detecção de anomalias em logs, geradores de senhas seguras e a instalação manual do MySQL.

### Artigos 61 a 66


**#61** AIOps na Prática: O Que a IA Já Faz em Operações Hoje
**#62** Gerando e Revisando Pipelines com IA
**#63** Detecção de Anomalias e Análise Inteligente de Logs com IA
**#64** Geração de Infraestrutura com IA
**#65** Geradores de Senhas Seguras: Implementações em 5 Linguagens de Programação
**#66** Instalação Manual do MySQL no Debian, Arch, Fedora e openSUSE


## Perguntas conceituais

**Q69** **Revisando o que a IA gerou**

Um assistente de IA gerou um *workflow* do GitHub Actions e um módulo Terraform para você. Que verificações são obrigatórias antes de aceitá-los, e por que "funcionou no primeiro teste" não basta?

**Q70** **Anomalia não é incidente**

Um sistema de AIOps detecta uma "anomalia" no volume de *logs* de erro. Por que nem toda anomalia deve virar alerta, e o que diferencia a detecção por limiar fixo da detecção por desvio do padrão?

### Q71 Instalando um banco sem contêiner

O artigo #66 instala o MySQL manualmente em quatro famílias de distribuições. Quais são os pacotes e os comandos de serviço no Debian e no Fedora, e que passo de segurança nenhuma instalação deve pular?

## Exercícios de código

### Q72 Detectando anomalias com z-score

Dada a contagem de erros por minuto ao longo de uma hora, calcule, para cada minuto, o *z-score* em relação à média e ao desvio padrão dos 15 minutos **anteriores**, e sinalize os minutos com z acima de 3. Compare com a hora inteira como referência, e com uma versão robusta, baseada na mediana e no desvio absoluto mediano (MAD).

### Q73 Um gerador de senhas no terminal

O artigo #65 gera senhas seguras em cinco linguagens. Escreva em Bash `gerar_senha TAMANHO`, que use `/dev/urandom` e garanta pelo menos uma minúscula, uma maiúscula, um dígito e um símbolo, recusando tamanhos menores que 12. Gere 1000 senhas e confira (sem imprimi-las) que todas cumprem as regras e que não há repetidas.

PARA IR ALÉM

# Desafios integradores

Projetos maiores, que juntam vários módulos. Não há uma única resposta certa: o gabarito mostra um caminho.

**D1**

### Uma calculadora de terminal em Bash

Escreva uma calculadora de terminal em Bash com sete operações: soma, subtração, multiplicação, divisão, potência (expoente inteiro), porcentagem ( `15 % 200` é 15% de 200) e raiz quadrada ( `raiz 2` ). Ela deve: (a) aceitar vírgula ou ponto como separador decimal; (b) guardar o último resultado em `ans` , utilizável na conta seguinte; (c) recusar divisão por zero, raiz de negativo, expoente fracionário, resultado infinito e texto que não é número com uma mensagem clara, **sem encerrar o programa**; (d) ter os comandos `hist` (histórico) e `sair` . Separe o núcleo ( `calc_nucleo.sh` , só funções, sem leitura de teclado) da interface ( `calc.sh` ). Como o Bash só faz contas com inteiros, use o `awk` para as contas decimais.

*Sua solução (arquitetura, decisões e código principal):*

**D2**

### Da máquina vazia ao deploy automatizado

Monte, de ponta a ponta, a entrega de uma aplicação web simples (a API de livros do caderno de PHP serve): o código no Git com *branches* protegidas; um pipeline que roda *lint*, testes e varredura de segredos, constrói a imagem e a publica num *registry* com a *tag* do *commit*; a infraestrutura de uma VM criada com Terraform e configurada com Ansible (Docker, usuário de deploy, *firewall*); o deploy automático com verificação de saúde e *rollback*; e um painel com latência, taxa de erros e um alerta baseado em SLO.

*Sua solução (arquitetura, decisões e código principal):*

**Bom estudo e bons exercícios!**

Estudar e realmente Aprender DevOps & Cloud · theknowledgeisle.com